Université Hassan 1er
Faculté des Sciences
et Techniques de
Settat

Commandes Linux

Réalisé par : Abdellah EZZATI

Sommaire

Chapitre 1

Les commandes Unix de base à connaître

---

man : Cette commande permet de donner une aide sur la commande demandée

Affiche les pages du manuel système. Chaque argument donné a man est généralement le nom d'un programme, d'un utilitaire, d'une fonction ou d'un fichier spécial.

Exemples d'utilisation :

- **man man**
  affiche les informations pour l'utilisation de man
- **man exports**
  décrit le contenu et la syntaxe du fichier **/etc/exports** pour les partages NFS

ls : affiche le contenu d'un répertoire

Équivalent MS-DOS/MS **Windows :** dir. Elle des options, les plus fréquentes sont :

- **-l** : Permet un affichage détaillé du répertoire (permissions d'accès, le nombre de liens physiques, le nom du propriétaire et du groupe, la taille en octets, et l'horodatage)
- **-h** : Associé avec **-l** affiche la taille des fichiers avec un suffixe correspondant à l'unité (K, M, G)
- **-a** : Permet l'affichage des fichiers et répertoires cachés (ceux qui commencent par un . (point))
- **-lct** : Permet de trier les fichiers et répertoires par date de modification décroissante

- Exemples d'utilisation :
  - **ls -a**
    affiche tous les fichiers et répertoires y compris les cachés du répertoire courant
  - **ls /etc/**
    affiche le contenu du répertoire /etc/
  - **lspci** ou **lsusb**
    affiche les périphériques PCI ou USB connectés.

Il existe également une commande **dir** quasi identique à la commande **ls**. Elle s'utilise de la même façon, avec les mêmes options, et les pages man (manuel) des 2 commandes sont d'ailleurs identiques. Seul l'affichage par défaut de la sortie est différent avec **dir** :

- par défaut la sortie n'est pas en couleur (il faut utiliser l'option **--color** pour obtenir une sortie en couleur avec dir) ;
- les caractères spéciaux tels que les espaces dans les noms des fichiers et dossiers sont précédés d'un caractère \ (backslash).

La commande « **ls -C -b** » produira une sortie identique à la commande **dir** employée sans options.

Modifier

cd : permet de changer le répertoire

Permet de se promener dans les répertoires. Exemples d'utilisation :

- **cd**
  permet de revenir au répertoire /home/utilisateur (identique à cd ~)
- **cd -**
  permet de revenir au répertoire précedent
- **cd ..**
  permet de remonter au répertoire parent (ne pas oublier l'espace contrairement à windows)
- **cd /**
  permet de remonter à la racine de l'ensemble du système de fichiers
- **cd /usr/bin/**
  se place dans le répertoire /usr/bin/

mv : déplace ou renomme un répertoire

Permet de déplacer ou renommer des fichiers et des répertoires. Ces Options les plus fréquentes :

- **-f** : Ecrase les fichiers de destination sans confirmation
- **-i** : Demande confirmation avant d'écraser
- **-u** : N'écrase pas le fichier de destination si celui-ci est plus récent

- Exemples d'utilisation :
  - **mv monFichier unRep/**
    Déplace *monFichier* dans le répertoire *unRep*
  - **mv unRep/monFichier .**
    Déplace le fichier *monFichier* du répertoire *unRep* là où on se trouve
  - **mv unRep monRep**
    Renomme *unRep* en *monRep*

cp : copie un fichier sur un autre

Permet de copier des fichiers ou des répertoires. Ces options les plus fréquentes :

- **-a** : Archive. Copie en gardant les droits, dates, propriétaires, groupes, etc.

- **-i** : Demande une confirmation avant d'écraser
- **-f** : Si le fichier de destination existe et ne peut être ouvert alors le détruire et essayer à nouveau
- **-R** ou **-r** : Copie un répertoire et tout son contenu, y compris les éventuels sous-répertoires
- **-u** : Ne copie que les fichiers plus récents ou qui n'existent pas
- **-v** : permet de suivre les copies réalisées en temps réel

- Exemples d'utilisation :
  - **cp monFichier sousrep/**
    Copie *monFichier* dans *sousrep*
  - **cp -r monRep/ ailleurs/**
    Copie le répertoire *monRep* (et ses éventuels sous-répertoires) vers *ailleurs* en créant le répertoire *ailleurs/monRep* s'il n'existe pas.

Mkdir : permet de créer un ou plusieurs répertoires

Elle des options et les plus fréquentes sont :

- **-p** : Crée les répertoires parents s'ils n'existent pas

- Exemples d'utilisation :
  - **mkdir photos**
    Crée le répertoire *photos*
  - **mkdir -p photos/2005/noel**
    Crée le répertoire *noel* et s'ils n'existent pas les répertoires *2005* et *photos*

rmdir : Supprime un répertoire

Supprime un répertoire (vide). Ces options les plus fréquentes :

- **-p** : Supprime les répertoires parents s'ils deviennent vides

- Exemples d'utilisation :
  - **rmdir LeRep**
    Supprime le répertoire *LeRep*

top : Montre la charge CPU

- La combinaison des touches [Majuscules + M] permet de classer en fonction de l'occupation de la mémoire.
- La combinaison des touches [Majuscules + P] classe en fonction de l'occupation du CPU.
- La combinaison des touches [Majuscules + W] permet de conserver ces préférences
- La touche [k] permet de tuer directement un processus en rentrant son PID
- La touche [q] permet de quitter le programme
- Options les plus fréquentes :
  - **-u** : affiche les processus pour un utilisateur donné
- Exemples d'utilisation :
  - **top**
  - **top -u root**

- 'q' pour quitter.
- Alternative : htop qui est plus complet et disponible dans les dépôts (htop)

pwd

- Équivalent MS-DOS/MS Windows : **chdir**
- Signification : *print working directory*
- Affiche le répertoire en cours

ln

- Signification : *link*
- Crée un lien (physique ou symbolique) vers un fichier (ou un répertoire)
- Options les plus fréquentes :
  - **-s** : Crée un lien symbolique (similaire au raccourci du monde Windows)
  - **-f** : Force l'écrasement du fichier de destination s'il existe
  - **-d** : Crée un lien sur un répertoire (uniquement en mode sudo ou root)
- Exemples d'utilisation :
  - **ln -s Rep1/Rep2/Monfichier MonLien**
    Crée un lien symbolique *MonLien* de *Rep1/Rep2/Monfichier* dans le répertoire où on se trouve
  - **ln Monfichier unRep/AutreNom**
    Crée un lien physique *AutreNom* de *Monfichier* dans le répertoire *unRep*
- Note :
  - Vérifiez que vous vous trouvez bien dans le répertoire dans lequel vous souhaitez créer le lien avant d'exécuter cette commande.

find

- Équivalent MS-DOS/MS Windows : **find**
- Signification : *rechercher*
- Permet de chercher des fichiers et éventuellement d'exécuter des commandes sur ceux-ci ; la recherche est *récursive* c'est-à-dire qu'elle concerne le répertoire de départ et toute sa descendance (sous-répertoires ainsi que toute leur descendance …)
- Options les plus fréquentes :
  - **-name** : Recherche d'un fichier par son nom
  - **-iname** : Même chose que **-name** mais insensible à la casse
  - **-type** : Recherche de fichier d'un certain type
  - **-atime** : Recherche par date de dernier accès
  - **-mtime** : Recherche par date de dernière modification
  - **-link** : Recherche du nombre de liens au fichier
  - **-user** : Recherche de fichiers appartenant à l'utilisateur donné
  - **-group** : Recherche de fichiers appartenant au groupe donné
- Actions les plus fréquentes :
  - **-exec** : Exécute la commande donnée aux fichiers trouvés
  - **-ok** : Même chose que **-exec** mais demande une confirmation
  - **-ls** : exécute la commande **ls** à chaque fichier trouvé
- Opérateurs les plus fréquents :

  - **-a** : Opérateur ET
  - **-o** : Opérateur OU
  - **!** ou **-not** : Opérateur NOT
- Exemples d'utilisation :

Placez-vous dans le répertoire à partir duquel la recherche *récursive* doit être effectuée et faites :

- **find monfichier***
  Recherche un fichier commençant par "monfichier"
- **find -name *monfichier*.ogg**
  Recherche un fichier contenant "monfichier" et ayant pour extention ".ogg"

- **find /home/ -name monfichier**
  Recherche le fichier *monfichier* dans toute la descendance de /home/
- **find . -name "*.c"**
  Recherche tous les fichiers ayant une extension *.c*
- **find . -mtime -5**
  Recherche les fichiers du répertoire courant qui ont été modifiés entre maintenant et il y a 5 jours
- **find /home/ -mtime -1 \! -type d**
  Recherche uniquement les fichiers (*! -type d* signifie n'était pas un répertoire) ayant été modifiés ces dernières 24h
- **find . ! -user root**
  Affiche tous les fichiers n'appartenant pas à l'utilisateur root
- **find . \(-name '*.wmv' -o -name '*.wma' \) -exec rm {} \;**
  Recherche et supprime tous les fichiers WMA et WMV trouvés
- **find . \(-type f -exec sudo chmod 664 "{}" \; \) , \(-type d -exec sudo chmod 775 "{}" \; \)**
  Modifie récursivement les droits en 664 sur les fichiers et en 775 sur les répertoires en une seule instruction

grep

- Équivalent MS-DOS/MS Windows : **find**
- Signification : *global regular expression print*
- Recherche une chaîne de caractères dans des fichiers (ou depuis la console si aucun fichier n'est indiqué) ; Souvent utilisé en filtre avec d'autres commandes.
- Options les plus fréquentes :
  - **-c** : Retourne le nombre de lignes au lieu des lignes elles mêmes
  - **-n** : Retourne les lignes préfixées par leur numéro
  - **-i** : Insensible à la casse
  - **-r** : Recherche récursivement dans tous les sous-répertoires ; On peut utiliser la commande **rgrep**
  - **-G** : Recherche en utilisant une expression rationnelle basique (option par défaut)
  - **-E** : Recherche en utilisant une expression rationnelle étendue ; On peut utiliser la commande **egrep**
  - **-F** : Recherche en utilisant une chaîne fixe ; On peut utiliser la commande **fgrep**
- Exemples d'utilisation :
  - **grep -n montexte monfichier**
    Retourne toutes les lignes ainsi que leur numéro où *montexte* apparait dans *monfichier*

locate

Son utilisation - très simple - est détaillée ici : recherche_ligne_commande

cat

- Équivalent MS-DOS/MS Windows : **type**
- Signification : *concatenate*
- Affiche le contenu d'un fichier
- Options les plus fréquentes :
  - **-n** : Affiche les numéros de ligne
  - **-v** : Affiche les caractères de contrôles
- Exemple d'utilisation :
  - **cat -n monFichier**
    Affiche *monFichier* en numérotant les lignes à partir de 1
  - créer un fichier texte contenant quelques lignes sans avoir recours à un éditeur :

more : Affiche un fichier page par page

- Options les plus fréquentes :
  - **-s** : Regroupe les lignes vides consécutives en une seule
  - **-f** : Ne coupe pas les lignes longues
- Exemple d'utilisation :
  - **more -sf monFichier**
    Affiche *monFichier* page par page en concaténant les lignes vides sans couper les lignes longues.

less

- Équivalent MS-DOS/MS Windows : **more**
- Signification : *less*
- Affiche un fichier en permettant la navigation, ainsi que certaines possibilités de vi(par ex: la recherche)
- Options les plus fréquentes :
  - **-e** ou **-E** : Quitte automatiquement la deuxième fois que la fin du fichier est atteinte, ou dès la première fois avec **-E**.
  - **-F** : Quitte automatiquement si le fichier tient sur le terminal.
  - **-m** ou **-M** : Prompt long a la **more**.
  - **-r** ou **-R** : Autorise les caractères spéciaux.
  - **-x** : Règle la taille des tabulations.
  - **-~** : ne comble pas les lignes vides par des ~
- Exemple d'utilisation :
  - **less -Emr~ monFichier**
    Affiche *monFichier* page par page avec un prompt long (affichage du pourcentage du fichier parcouru) en affichant les caractères spéciaux sans combler les lignes vides par des ~

Chapitre 2

Commandes avancées

Dans ce chapitre on va étudier certaines commandes avancées

Chown Change le propriétaire et le groupe propriétaire d'un fichier

Cette commande change le propriétaire et le groupe propriétaire d'un fichier

- Options les plus fréquentes :
  - **-R** : Modifie récursivement un répertoire et tout ce qu'il contient
- Exemples d'utilisation :
  - **chown autreUtilisateur MonFichier**
    Change le propriétaire de *MonFichier* en *autreUtilisateur*
  - **chown -R lui:nous monRep**
    Change le propriétaire en *lui* et le groupe propriétaire en *nous* du répertoire *monRep* ainsi que tout ce qu'il contient

chgrp : Change le groupe propriétaire d'un fichier

Change le groupe propriétaire d'un fichier

Options les plus fréquentes de cette commande:

- **-R** : Change récursivement un répertoire et tout ce qu'il contient
- **-h** : Change le groupe propriétaire d'un lien symbolique et seulement lui (ne touche pas à la destination du lien)
- **-L** : Si fournie avec **R** , change le groupe propriétaire d'un répertoire et des fichiers qu'il contient s'il est pointé par un lien symbolique rencontré lors de l'exécution

- Exemples d'utilisation :
  - **chgrp unGroupe MonFichier**
    Change le groupe propriétaire du fichier *MonFichier* en *unGroupe*
  - **chgrp -R unGroupe monRep**
    Change le groupe propriétaire du répertoire *monRep* ainsi que tout ce qu'il contient en *unGroupe*

free

- Signification : *mémoire libre*
- Affiche la mémoire disponible / utilisée du système
- Options les plus fréquentes :
  - **-b** : Affiche la mémoire en bytes
  - **-k** : Affiche la mémoire en kilo octet

  - **-m** : Affiche la mémoire en méga octet
  - **-g** : Affiche la mémoire en giga octet
  - **-s** : Spécifie le délai de réaffichage de la mémoire
  - **-t** : Affiche la ligne des totaux
- Exemples d'utilisation :
  - **free -m -s 5**
    Affiche la mémoire du système en méga octet toutes les 5 secondes

mount

- Monter un système de fichiers
- Options les plus fréquentes :
  - **-a** : Monter tous les systèmes de fichier déclarés dans le fichier */etc/fstab*
  - **-t** : Précise le type de fichier à monter
  - **-o** : Ajouter une option. Options adjointe à **-o** les plus fréquentes :
  - **auto** : Permet d'être monté par **-a**
  - **async** : Les entrées/sorties sur le système de fichiers seront asynchrones
  - **defaults** : Utilise les options **rw**, **suid**, **dev**, **exec**, **auto**, **nouser**, et **async**.
  - **dev** : Interprète les fichiers spéciaux de périphériques du système présent dans */dev/*
  - **exec** : Permet l'exécution de fichiers binaires du système monté
  - **noauto** : Empêche d'être monté avec **-a**
  - **nodev** : Ne pas interpréter les fichiers spéciaux de périphériques du système
  - **noexec** : Empêche l'exécution de fichiers binaires du système monté
  - **nouser** : Ne pas autoriser d'autres utilisateurs que root (ou sudo) à monter le système de fichiers (comportement par défaut)
  - **ro** : Monte le système en lecture seule
  - **rw** : Monte le système en lecture et écriture
  - **suid** : Prend en compte les bits SetUID ou SetGID du système monté
  - **user** : Permet aux utilisateurs ordinaires à monter et démonter le système de fichiers (implique **noexec**, **nosuid**, et **nodev** sauf si surchargées)
- Exemples d'utilisation :
  - **mount**
    Liste tous les systèmes de fichiers actuellement montés
  - **mount -a**
    Monte tous les systèmes de fichiers déclarés dans le fichier */etc/fstab*
  - **mount /mnt/maPartion**
    Monte le système de fichiers ad-hoc déclarés dans le fichier */etc/fstab*
  - **mount -t iso9660 monFichier.iso /mnt/monIso -o loop**
    Monte dans un *périphérique boucle* (loop) le fichier iso *monFichier.iso* dans le répertoire */mnt/monIso*
  - **mount -t vfat -o defaults,rw,user,umask=022,uid=1000 /dev/sda1 /mnt/Mondisk/**
    Monte un disque dur USB (*/dev/sda1*) formaté en FAT32 (*-t vfat*) en lecture écriture (*rw*) dans le répertoire */mnt/Mondisk/* ; tous les utilisateurs peuvent le démonter (*user*), les droits d'exécution (*uid=1000*) sont fixés à l'utilisateur ayant l'UID 1000 (sous Ubuntu, l'uid 1000 correspond au premier utilisateur créé) et la création d'un fichier s'effectuera avec les permissions 644 (rw-r---r--) et pour un répertoire 755 (rwxr-xr-x) (*umask 022*)

umount

- Démonte un système de fichiers
- Options les plus fréquentes :
  - **-a** : Démonte tous les systèmes de fichiers présents dans */etc/mtab*
  - **-d** : Si le système monté est un périphérique *loop*, libérer le périphérique.
  - **-f** : Forcer le démontage
  - **-r** : Si impossible de démonter, monter en lecture seule
- Exemples d'utilisation :
  - **umount /mnt/Mondisk**
    Démonte le système de fichiers monté dans */mnt/Mondisk*
  - **umount -f /dev/cdrom**
    Force le démontage du périphérique CDROM
  - **umount -d /mnt/monIso**
    Démonte et libère le périphérique loop
  - **umount -a**
    Démonte tous les systèmes de fichiers montés (à l'exception de */proc*) ; ne sert que lorsque l'on veut redémarrer ou éteindre sa machine manuellement et proprement.

sudo

- Permet d'exécuter des commandes en tant qu'un autre utilisateur, donc avec d'autres privilèges que les siens.
- Options les plus fréquentes :
  - **-s** : Importe les variables d'environnement du shell
  - **-k** : Lorsque l'on utilise **sudo**, il garde en mémoire le mot de passe ; cette option déconnecte l'utilisateur et forcera à redemander un mot de passe si **sudo** est exécuté avant le timeout défini.
- Exemples d'utilisation :
  - **sudo reboot**
    Lance la commande **reboot** avec les droits de l'utilisateur root
- Ressources :
  - sudo
  - Voir aussi la commande **visudo**

ps Affiche les processus en cours

- Options les plus fréquentes :
  - **-u** : Affiche les processus de l'utilisateur qui exécute la commande
  - **-au** : Affiche les processus de tous les utilisateurs
  - **-aux** : Affiche l'intégralité des processus du système. Équivalent à **ps -A**
  - **-faux** : Affiche tous les processus du système en les regroupant par enchaînement d'exécution.
- Exemples d'utilisation :
  - **ps -u**
    Tous les processus de l'utilisateur courant
  - **ps -aux**
    Tous les processus en cours

pensez à utiliser avec grep pour limiter la liste : ps -aux | grep tuxpaint ne vous retournera que les processus contenant tuxpaint

kill / killall

- Permet d'envoyer un signal à un processus ; **kill** ne comprend que les PID (Process Identifier, numéro d'ordre du processus), **killall** quant à lui comprend le nom du processus.
- Options les plus fréquentes :
  - **-s** : Indique quel signal *s* à envoyer au processus ; Le signal peut être identifié soit par son nom (exemple : SIGTERM) soit par son numéro (exemple : 9) ; Cette option peut être remplacée par le numéro du signal : **-s 9** est équivalent à **-9**.
  - **-l** : Affiche la liste des signaux connus.
- Les signaux les plus courants sont :
  HUP signal **1** : signal de fin d'exécution ou le processus doit relire son fichier de configuration.
  TERM signal **15** : Le signal Terminate indique à un processus qu'il doit s'arrêter.
  KILL signal **9** : Le signal Kill indique au système qu'il doit arrêter un processus qui ne répond plus.
- Exemples d'utilisation :
  - **kill -15 14774** : Envoie le signal 15, ou TERM, au processus ayant le numéro 14774 ce qui a pour effet de **terminer** *proprement* le processus.
  - **kill -9 7804** : Envoie le signal 9, ou KILL, au processus ayant le numéro 7804 ce qui a pour effet de **tuer** le processus.
  - **killall -TERM firefox-bin** : Envoie le signal TERM, ou 15, au processus firefox-bin ce qui a pour effet de le fermer.
- Il est généralement conseillé de lancer des signaux de faible importance avant de lancer la grosse artillerie. En pratique, tester dans l'ordre et deux fois chacune de ces commandes :

Chapitre 3

Script Shell

Sous Unix, on appelle *shell* l'interpréteur de commandes qui fait office d'interface entre l'utilisateur et le système d'exploitation. Les shells sont des interpréteurs : cela signifie que chaque commande saisie par l'utilisateur (ou lue à partir d'un fichier) est syntaxiquement vérifiée puis exécutée.

Il existe de nombreux shells qui se classent en deux grandes familles :

- la famille *C shell* (ex : **csh**, **tcsh**)
- la famille *Bourne shell* (ex : **sh**, **bash**, **ksh**, **dash**).

zsh est un shell qui contient les caractéristiques des deux familles précédentes. Néanmoins, le choix d'utiliser un shell plutôt qu'un autre est essentiellement une affaire de préférence personnelle ou de circonstance. En connaître un, permet d'accéder aisément aux autres. Lorsque l'on utilise le système *GNU/Linux* (un des nombreux systèmes de la galaxie Unix), le shell par défaut est **bash** (*Bourne Again SHell*). Ce dernier a été conçu en 1988 par Brian Fox dans le cadre du projet GNU [2]. Aujourd'hui, les développements de **bash** sont menés par Chet Ramey.

Un shell possède un double aspect :

- un aspect *environnement de travail*
- un aspect *langage de programmation*.

Un environnement de travail

En premier lieu, un shell doit fournir un environnement de travail agréable et puissant. Par exemple, **bash** permet (entre autres) :

- le rappel des commandes précédentes (gestion de l'historique) ; cela évite de taper plusieurs fois la même commande
- la modification en ligne du texte de la commande courante (ajout, retrait, remplacement de caractères) en utilisant les commandes d'édition de l'éditeur de texte **vi** ou **emacs**
- la gestion des travaux lancés en arrière-plan (appelés *jobs*) ; ceux-ci peuvent être démarrés, stoppés ou repris suivant les besoins
- l'initialisation adéquate de variables de configuration (chaîne d'appel de l'interpréteur, chemins de recherche par défaut) ou la création de raccourcis de commandes (commande interne **alias**).

Illustrons cet ajustement de configuration par un exemple. Le shell permet d'exécuter une commande en mode interactif ou bien par l'intermédiaire de fichiers de commandes (*scripts*). En mode interactif, **bash** affiche à l'écran une *chaîne d'appel* (appelée également *prompt* ou *invite*), qui se termine par défaut par le caractère **#** suivi d'un caractère **espace** pour l'administrateur système (utilisateur **root**) et par le caractère **$** suivi d'un caractère **espace** pour les autres utilisateurs. Cette chaîne d'appel peut être relativement longue.

Celle-ci est constituée du nom de connexion de l'utilisateur (*sanchis*), du nom de la machine sur laquelle l'utilisateur travaille (*jade*) et du chemin absolu du répertoire courant de l'utilisateur (*/bin*). Elle indique que le shell attend que l'utilisateur saisisse une commande et la valide en appuyant sur la touche **entrée**. Bash exécute alors la commande puis réaffiche la chaîne d'appel.

Si l'on souhaite raccourcir cette chaîne d'appel, il suffit de modifier la valeur de la variable prédéfinie du shell **PS1** (*Prompt Shell 1*).

La nouvelle chaîne d'appel est constituée par le caractère **$** suivi d'un caractère espace.

Un langage de programmation

Les shells ne sont pas seulement des interpréteurs de commandes mais également de véritables langages de programmation. Un shell comme **bash** intègre :

- les notions de *variable*, *d'opérateur arithmétique*, de *structure de contrôle*, de *fonction*, présentes dans tout langage de programmation classique, mais aussi
- des opérateurs spécifiques (ex : **|**, **&**)

```
$ a=5  => affectation de la valeur 5 à la variable a
$
$ echo $((a +3 ))  => affiche la valeur de l'expression a+3
8
$
```

L'opérateur **|**, appelé *tube*, est un opérateur caractéristique des shells et connecte la sortie d'une commande à l'entrée de la commande suivante.

```
$ lpstat -a
HP-LaserJet-2420 acceptant des requêtes depuis jeu. 14 mars 2013 10:38:37 CET
HP-LaserJet-P3005 acceptant des requêtes depuis jeu. 14 mars 2013 10:39:54 CET
$
$ lpstat -a | wc -l
2
$
```

La commande unix **lpstat** permet de connaître les noms et autres informations relatives aux imprimantes accessibles. La commande unix **wc** munie de l'option **l** affiche le nombre de lignes qu'elle a été en mesure de lire.

Atouts et inconvénients des shells

L'étude d'un shell tel que **bash** en tant que langage de programmation possède plusieurs avantages :

- c'est un langage interprété : les erreurs peuvent être facilement localisées et traitées ; d'autre part, des modifications de fonctionnalités sont facilement apportées à l'application sans qu'il soit nécessaire de recompiler et faire l'édition de liens de l'ensemble
- le shell manipule essentiellement des chaînes de caractères : on ne peut donc construire des structures de données complexes à l'aide de pointeurs, ces derniers n'existant pas en shell. Ceci a pour avantage d'éviter des erreurs de typage et de pointeurs mal gérés. Le développeur raisonne de manière uniforme en termes de chaînes de caractères
- le langage est adapté au prototypage rapide d'applications : les tubes, les substitutions de commandes et de variables favorisent la construction d'une application par assemblage de commandes préexistantes dans l'environnement Unix
- c'est un langage « glu » : il permet de connecter des composants écrits dans des langages différents. Ils doivent uniquement respecter quelques règles particulièrement simples. Le composant doit être capable :
    - de lire sur l'entrée standard,
    - d'accepter des arguments et options éventuels,
    - d'écrire ses résultats sur la sortie standard,
    - d'écrire les messages d'erreur sur la sortie standard dédiée aux messages d'erreur.

Les conditions

La syntaxe en bash est la suivante :

```
if [ test ]
then
        echo "C'est vrai"
fi
```

```
#!/bin/bash

nom="Bruno"

if [ $nom = "Bruno" ]
then
        echo "Salut Bruno !"
fi
```

Notez aussi que vous pouvez tester deux variables à la fois dans le `if` :

```
#!/bin/bash

nom1="Bruno"
nom2="Marcel"

if [ $nom1 = $nom2 ]
then
        echo "Salut les jumeaux !"
fi
```

Comme ici `$nom1` est différent de `$nom2`, le contenu du `if` ne sera pas exécuté. Le script n'affichera donc rien.

Sinon

Si vous souhaitez faire quelque chose de particulier quand la condition n'est **pas** remplie, vous pouvez rajouter un `else` qui signifie « sinon ».

En français, cela s'écrirait comme ceci :

```
SI test_de_variable
ALORS
------> effectuer_une_action
SINON
------> effectuer_une_action
FIN SI
if [ test ]
then
        echo "C'est vrai"
else
        echo "C'est faux"
fi
```

Reprenons notre script de tout à l'heure et ajoutons-lui un `else` :

```
#!/bin/bash

nom="Bruno"
```

```
if [ $nom = "Bruno" ]
then
        echo "Salut Bruno !"
else
        echo "J'te connais pas, ouste !"
fi
```

Chapitre 4

Fonction en Bash

Définition d'une fonction

Le shell **bash** propose plusieurs syntaxes pour définir une fonction. Nous utiliserons celle-ci :

function *nom_fct*

{

suite_de_commandes

}

nom_fct spécifie le nom de la fonction. Le corps de celle-ci est *suite_de_commandes*.

Pour appeler une fonction, il suffit de mentionner son nom.

Comme pour les autres commandes composées de **bash**, une fonction peut être définie directement à partir d'un shell interactif.

```
$ function f0
> {
> echo Bonjour tout le monde !
> }
$
$ f0    => appel de la fonction f0
Bonjour tout le monde !
$
```

Les mots réservés **function** et **}** doivent être les premiers mots d'une commande pour qu'ils soient reconnus. Sinon, il suffit de placer un caractère **point-virgule** avant le mot-clé :

function *nom_fct*

{ *suite_de_commandes* **;}**

La définition d'une fonction « à la C » est également possible :

function *nom_fct* **{**

suite_de_commandes **}**

L'exécution d'une fonction s'effectue dans l'environnement courant, autorisant ainsi le partage de variables.

```
$ c=Coucou
$
$ function f1
> {
> echo $c    => utilisation dans la fonction d'une variable externe c
> }
$
$ f1
Coucou
$
```

Les noms de toutes les fonctions définies peuvent être listés à l'aide de la commande : **declare -F**

```
$ declare -F
declare -f f0
declare -f f1
$
```

Les noms et corps de toutes les fonctions définies sont affichés à l'aide de la commande : **declare -f**

```
$ declare -f
f0 ()
{
  echo Bonjour tout le monde !
}
f1 ()
{
  echo $c
}
$
```

Pour afficher le nom et corps d'une ou plusieurs fonctions : **declare -f** *nomfct* ...

```
$ declare -f f0
f0 ()
{
```

```
 echo Bonjour tout le monde !
}
$
```

Une définition de fonction peut se trouver en tout point d'un programme shell ; il n'est pas obligatoire de définir toutes les fonctions en début de programme. Il est uniquement nécessaire que la définition d'une fonction soit faite avant son appel effectif, c'est-à-dire avant son exécution :

function *f1*

{ ... **;}**

suite_commandes1

function *f2*

{ ... **;}**

suite_commandes2

Dans le code ci-dessus, *suite_commandes1* ne peut exécuter la fonction *f2* (contrairement à *suite_commandes2*). Cela est illustré par le programme shell*appelAvantDef* :

```
appelAvantDef
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
          # !/bin/bash

          echo Appel Avant definition de fct
          fct       # fct non definie

          function fct
          {
          echo Execution de : fct
          sleep 2
          echo Fin Execution de : fct
          }
          echo Appel Apres definition de fct
          fct       # fct definie
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Son exécution se déroule de la manière suivante :

```
$ appelAvantDef
```

```
Appel Avant definition de fct
./appelAvantDef: line 4: fct: command not found
Appel Apres definition de fct
Execution de : fct
Fin Execution de : fct
$
```

Lors du premier appel à la fonction *fct*, celle-ci n'est pas définie : une erreur d'exécution se produit. Puis, le shell lit la définition de la fonction *fct* : le deuxième appel s'effectue correctement.

Contrairement au programme précédent, dans le programme shell *pingpong*, les deux fonctions *ping* et *pong* sont définies avant leur appel effectif :

pingpong

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        #!/bin/bash

        function ping
        {
        echo ping
        if (( i > 0 ))
          then
            ((i--))
            pong
        fi
        }
        function pong
        {
        echo pong
        if (( i > 0 ))
          then
            ((i--))
            ping
        fi
        }
        declare -i i=4

        ping    # (1) ping et pong sont definies
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Au point **(1)**, les corps des fonctions *ping* et *pong* ont été lus par l'interpréteur de commandes **bash** : *ping* et *pong* sont définies.

```
$ pingpong
ping
pong
ping
pong
ping
$
```

Suppression d'une fonction

Une fonction est rendue indéfinie par la commande interne : **unset -f** *nomfct* ...

```
$ declare -F
declare -f f0
declare -f f1
$
$ unset -f f1
$
$ declare -F
declare -f f0    => la fonction f1 n'existe plus !
$
```

Arguments d'une fonction

Les arguments d'une fonction sont référencés dans son corps de la même manière que les arguments d'un programme shell le sont : **$1** référence le premier argument, **$2** le deuxième, etc., **$#** le nombre d'arguments passés lors de l'appel de la fonction.

Le paramètre spécial **$0** n'est pas modifié : il contient le nom du programme shell.

Pour éviter toute confusion avec les paramètres de position qui seraient éventuellement initialisés dans le code appelant la fonction, la valeur de ces derniers est sauvegardée avant l'appel à la fonction puis restituée après exécution de la fonction.